An Evaluation of Interactive Test-Driven Labs with WebIDE in CS0

David S. Janzen, John Clements, Michael Hilton
California Polytechnic State University
San Luis Obispo, California 93407
{djanzen,clements,hilton}@calpoly.edu

***Abstract*—WebIDE is a framework that enables instructors to develop and deliver online lab content with interactive feedback. The ability to create lock-step labs enables the instructor to guide students through learning experiences, demonstrating mastery as they proceed. Feedback is provided through automated evaluators that vary from simple regular expression evaluation to syntactic parsers to applications that compile and run programs and unit tests. This paper describes WebIDE and its use in a CS0 course that taught introductory Java and Android programming using a test-driven learning approach. We report results from a controlled experiment that compared the use of dynamic WebIDE labs with more traditional static programming labs. Despite weaker performance on pre-study assessments, students who used WebIDE performed two to twelve percent better on all assessments than the students who used traditional labs. In addition, WebIDE students were consistently more positive about their experience in CS0.**

I. Introduction

WebIDE is a framework for creating a variety of labs that provide rich, rapid feedback to students as they learn new concepts and gain or practice new skills. WebIDE provides a scalable, open, and distributed infrastructure for lab authors to create, host, and render labs, using their own or provided automated evaluators. Figures 1 and 2 give examples of steps in existing WebIDE labs. Lab authors can establish dependencies between steps, requiring that students successfully complete a step before proceeding in the lab. Alternatively, we have set up WebIDE labs as a programming playground with a single step containing editor windows for writing code and corresponding unit tests, enabling a student to write simple programs completely in the web, even with compiled languages like C or Java.

At least twenty-four labs along with at least fourteen automated evaluators have already been created for introductory programming students learning C, Java, Android, Python, and Ruby. Instructors can write their own labs (or modify existing labs) using the WebIDE XML specification, then host the labs on their own web sites, pointing WebIDE to the lab files in order to parse, render, and run their own labs. Similarly, instructors can use provided automated evaluators to process student submissions and provide meaningful feedback in their labs, or they can write their own automated evaluators and host them on any internet-connected server. A video demo of WebIDE is available at http://www.web-ide.org/demo, and links to WebIDE labs organized into courses are available at http://web-ide.org.

WebIDE was developed and evaluated through an NSF CCLI Type 1 award with additional support from Google and Amazon. This paper will highlight key aspects of WebIDE including its support for Test-Driven Learning, along with its unique architecture for incorporating automated evaluators. Evaluation results from a semi-controlled experiment will be reported.

II. Test-Driven Learning

A key component of WebIDE's architecture is Test-Driven Learning (TDL) [24]. Closely related to Test-Driven Development [6] (TDD), TDL uses the construction of test cases to drive the learning process. In particular, TDL observes that students who write small test cases before the corresponding implementation tend to understand their goal before getting tangled up in code. TDL-based WebIDE labs require the students to write examples and/or test cases, and *check these test cases against instructor code*, before students tackle the more bulky task of writing the code itself. Figure 2 demonstrates an example of this approach.

Currently, TDD is becoming a widespread software engineering best practice. Previous studies indicate benefits from applying TDD, but note challenges of actually getting fledgling programmers to write code in a test-first manner[25]. Prior studies have shown that TDL can be applied in entry level classes without removing current course material, and that students who try TDL, tend to like TDL[24]. However, prior to WebIDE, finding ways to enforce a test-driven approach with beginning programmers has proven to be elusive [9].

The inspiration for WebIDE was the desire to require students to demonstrate understanding of a problem by writing examples and tests, prior to solving a problem. The goal was to instill software engineering best practices from the beginning of learning to program. Several secondary benefits were quickly discovered with WebIDE. In particular, WebIDE provided the opportunity to point students to a super-simple, web-only programming environment on day one. Students are able to delay the learning curve associated with editors, compilers, and integrated development environments, and concentrate on programming and problem solving from the very beginning.

WebIDE is not restricted to TDL, or even to computer programming for that matter. WebIDE provides an infrastructure designed so that anyone can create new or modify existing

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Software Engineering in Education

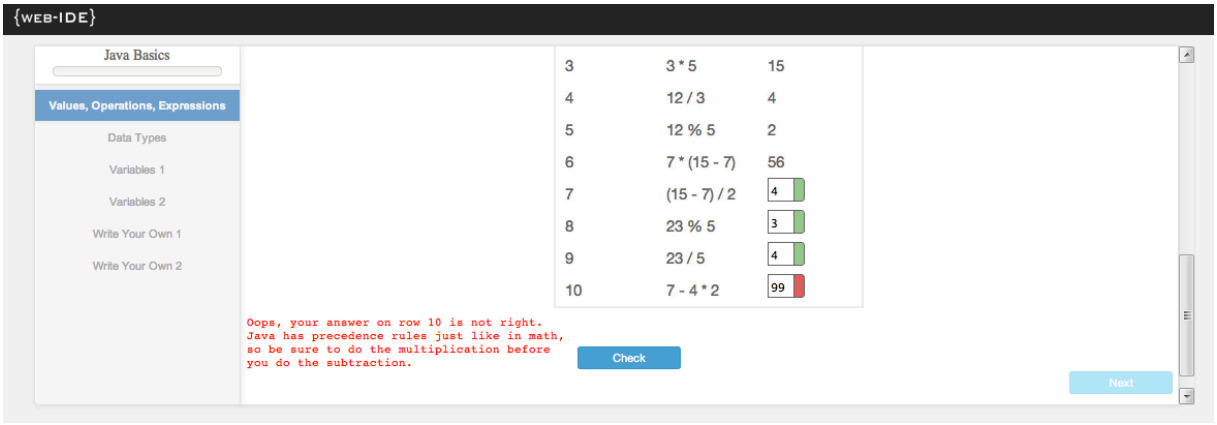

Fig. 1. WebIDE lab with sample error feedback

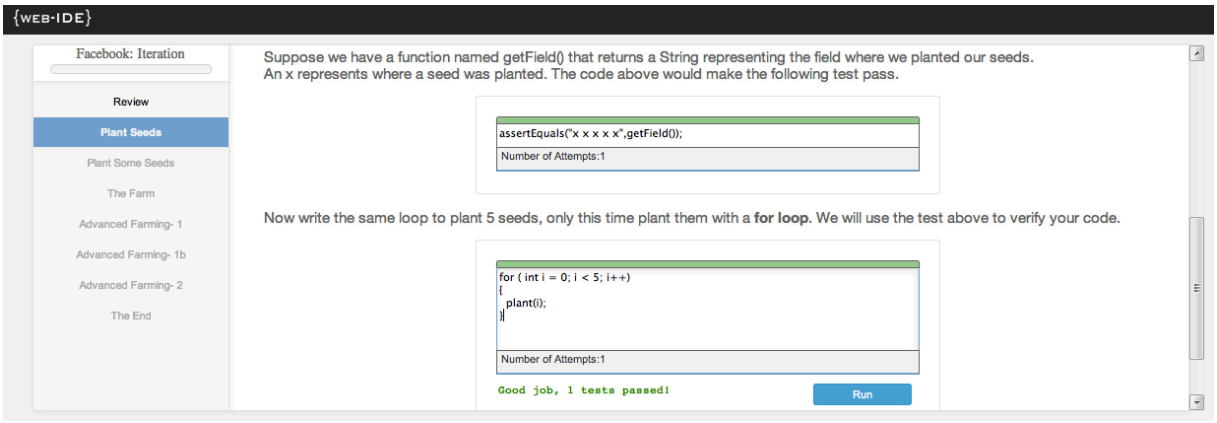

Fig. 2. Students create JUnit tests in WebIDE to demonstrate problem understanding

labs and evaluators—written in any language and providing customized error messages—that help teach a wide range of concepts, languages, or software engineering techniques.

III. WebIDE Architecture

WebIDE uses Google Web Toolkit (GWT) and is currently deployed on Amazon's EC2 cloud platform, although it has also been hosted on Google's App Engine in the past. The labs and automated evaluators can be located on any web-accessible host.

Figure 3 illustrates the general architecture of the system. The solid lines represent HTTP connections. The dashed line indicates an implicit dependency; specifically, the lab supplied by the Lab Source embeds within it URLs that allow WebIDE to locate the evaluators.

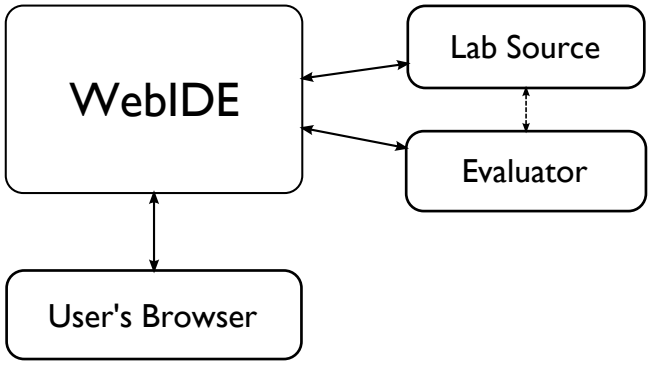

Fig. 3. WebIDE Architecture

The WebIDE architecture is focused on extensibility. Lab specifications are written in a well-defined XML language, so that labs may be edited and contributed by third parties. Additionally, the presentation of the lab is completely decoupled from the evaluator using a service-oriented architecture (SOA) where URLs identify evaluators.

To be more concrete, suppose an instructor wants to create a lab on FORTRAN/77. The instructor first formulates the text of the lab, using an XHTML subset. Then, (unless there's an existing FORTRAN/77 evaluator floating around), the instructor writes a script that couples a FORTRAN evaluator with the given submission and reports the result. Finally, the instructor embeds the URL of the evaluator within the lab itself. Later FORTRAN labs can re-use the instructor's existing evaluator.

In order to enforce structure and prevent ad-hoc extension, labs are written using a XML language defined using the Relax NG specification language. So, for instance, the lab is specified to contain a name, an optional description, and zero or more

steps:

```
start = element lab {
  attribute name { text },
  element description { text }?,
  step*
}
```

Jing [28] is used to validate a lab's XML. The parsing phase then maps that XML to a GWT page containing user entry fields.

The evaluator associated with a given lab step is responsible for determining the correctness of student entries. Since the evaluators can be hosted on any server by any author, an interface is supplied for communication between the engine and the evaluator using HTTP and encoding the request/response in JSON. Therefore, any language that can receive an HTTP request and send an HTTP response can be used to implement an evaluator. In fact, external evaluators are currently written in Java, PHP, and Racket.

The success of a tutor such as WebIDE depends crucially on the ability to deliver helpful error messages, and not a simple "success" or "failure." Accordingly, the JSON format for the evaluator's response includes a message. In future versions of the protocol, we anticipate including source highlighting information along with the message.

WebIDE supports both external and internal evaluators. External evaluators are invoked via JSON requests between machines. Internal evaluators are hosted within the WebIDE engine, and may therefore be faster and more reliable.

IV. WebIDE Labs

Figure 4 displays an example of a step in a lab that contains both JUnit tests and Java source code. In this example, the student forgot a set of parenthesis in the fToC method (should return (f - 32) * 5 / 9;). The two segments are highlighted in red because the tests are failing against the code. In this case, the student sees both the tests and the code. The lab author could hide the tests and ask the student to write the code, or hide the code and ask the students to write the tests. The lab author could even have a mix of student and instructor written hidden tests.

In this step, the lab author has elected to use the classUnitTest Java Evaluator. This is an external evaluator that runs a set of JUnit tests against code, returning "success" if the tests pass, and "failure" with the test results if they failed. The relevent portion of the corresponding XML lab file is shown in the class.xml listing.

Notice in Figure 4 that the last portion of the URL contains the URL of the XML file (http://www.csc.calpoly.edu/∼djanzen/courses/123F11/labs/class.xml). This open design allows any lab author to host their own labs, yet have them rendered by WebIDE. Furthermore, as the XML listing shows in the evaluator tag, lab authors can use provided evaluators that are hosted on WebIDE servers, or they can point to any evaluator available on the web.

V. Automated Evaluators

In the WebIDE framework, students provide answers to questions in web forms. Each answer is then sent to one or more "evaluators" to provide feedback. Effective and helpful feedback is the linchpin of a successful WebIDE interaction, and a variety of evaluators and evaluator families for use in different situations are already provided. The WebIDE framework accommodates the use of multiple evaluators for a single question box. Evaluators can reside on any server, but the provided evaluators are currently hosted on Amazon EC2 instances behind a load balancer, providing flexibility for handling inconsistent demand. We present two general categories of automated evaluators: syntactic and semantic.

A. Syntactic Evaluators

The "syntactic evaluators" are those that evaluate student code at a syntactic level, by considering it either as a stream of characters or as an abstract syntax tree. The simplest evaluators are those that simply perform textual comparison. Regular expressions provide a reasonably flexible way of performing such comparisons, and WebIDE includes a regexp-based evaluator. To take the simplest possible example, a lab asking a student to enter a positive integer might deliver the student's answer to the regexp evaluator matching the PERL regexp `^\s*[1-9]\d*\*$` .

For more complex problems, a typical syntactic evaluator is one based on a parser for the given language. The WebIDE team has developed a syntactic evaluator for C that parses student input and compares the resulting tree to one obtained by parsing an expected answer. This kind of evaluator has at least three advantages over one based on simple textual comparison. First, it has the advantage that it is insensitive to whitespace, the presence or absence of optional delimiters and parentheses, etc. Second, it can provide source code positions and source code highlighting along with its errors. Finally, it can abstract over certain observational equivalences; for instance, for terms A and B, the programs $A + B$ and $B + A$ may be considered equivalent. This assumes the language in question is purely functional, or is restricted to a purely functional subset, or that the sub-programs A and B can be statically analyzed to ensure they do not have side effects.

B. Semantic Evaluators

The "semantic evaluators" are those that consider the evaluated meaning of the student code, rather than its syntactic form. For instance, suppose a student is asked to produce a function called $smaller$ that accepts two numbers and returns the smaller of the two. A semantic evaluator would evaluate the student code and then apply it to one or more test inputs, verifying that it produces the expected output. The WebIDE team has produced semantic evaluators for a variety of Java problems, and general purpose semantic evaluators for C, Java, Python, and Ruby.

Semantic evaluators have advantages and disadvantages. On the one hand, they are more robust, in the sense that they can accept answers that a syntactic evaluator would not. On

Listing 1. class.xml

```
<step name="Static Methods" buttonName="Check">
<dependency stepName="Class Basics 2"/>
...omitted text introducing the topic...
<br/>
The first two tests are completed for you. Your job is to write testfToC2 and testcToF2.
<segment width="600" id="TCTests" height="300">
import org.junit.Test;
import junit.framework.TestCase;
public class TestTempConverter extends TestCase {
  @Test
  public void testfToC1() {
        assertEquals(19, TempConverter.fToC(67));
  }
  ...code omitted for brevity...
}
</segment>
Now write the code to make the tests above pass.
Be sure to define everything including the class with both methods.
Click the button at the bottom to run your tests above on your code below.
<segment width="600" id="TCCode" height="100">
public class TempConverter {
        public static long fToC(int f) {
                //replace this comment with the body of the fToC method
        }
        //replace this comment with the cToF method
}
</segment>
<evaluator name="Static Evaluator" labid="Unit Test Lab"
        href="http://evaluators.web-ide.org:8080/JavaEvaluators/classUnitTest">
  <arg>
    <name>class</name>
    <value>@TCCode</value>
  </arg>
  <arg>
    <name>testClass</name>
    <value>@TCTests</value>
  </arg>
  <arg>
    <name>smessage</name>
    <value>Good job, $testTotal test(s) passed!</value>
  </arg>
  <arg>
    <name>fmessage</name>
    <value>I'm sorry, $failTotal test failed... $failedTest</value>
  </arg>
  <segid>
    <id>TCCode</id>
  </segid>
  <segid>
    <id>TCTests</id>
  </segid>
</evaluator>
</step>
```

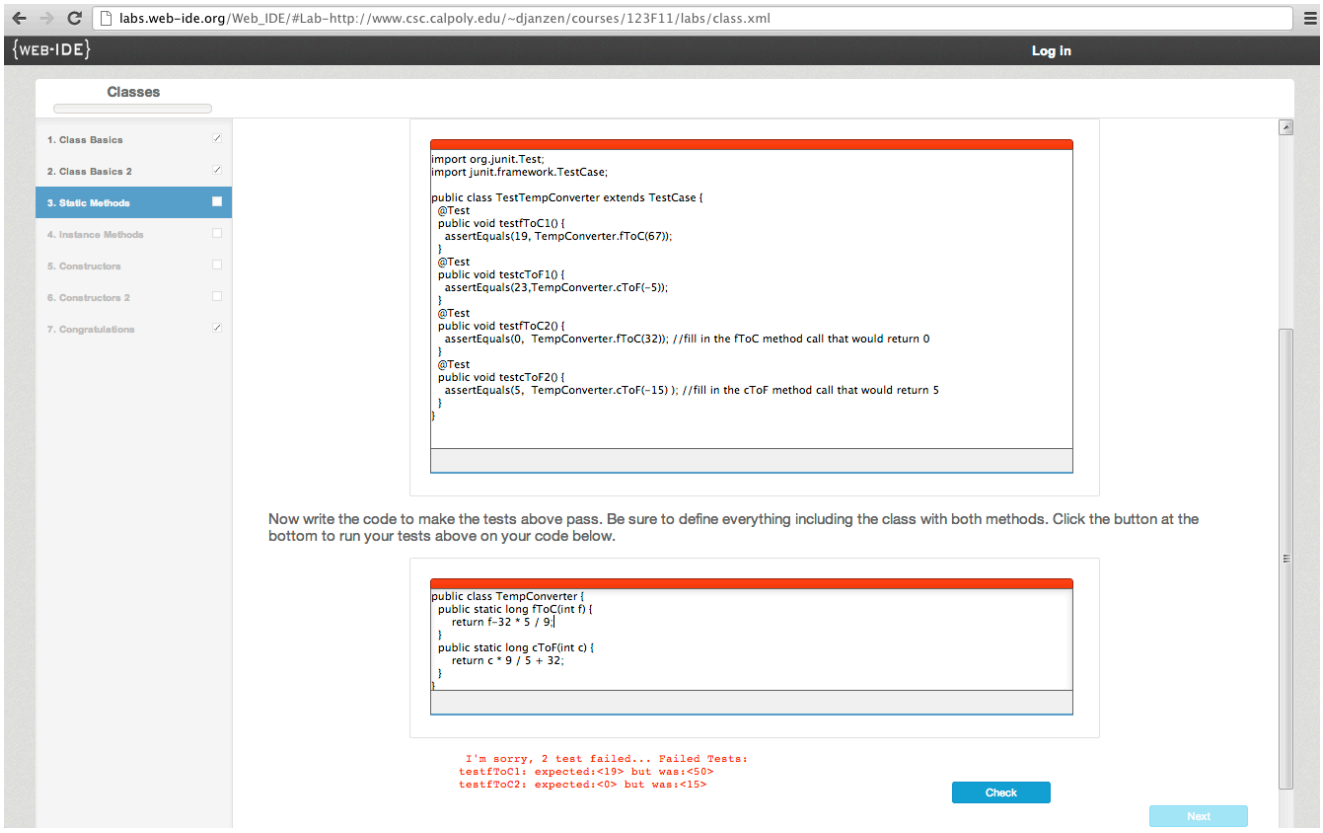

Fig. 4. Sample Step with Failing Tests in WebIDE

the other hand, regarded as provers of correctness, they are unsound; the halting problem demonstrates that no evaluator can decide for all problems whether an arbitrary program behaves correctly on all inputs.

An important subcategory of semantic evaluators are those that examine student-written test cases. In this case, the student's test case is run against the lab-writer's function. Test cases that do not correctly predict the output of the function do not succeed. More interestingly, a set of student test cases can also be evaluated for coverage; do they exercise all of the instructor's code?

VI. Evaluation

Three studies were conducted to evaluate WebIDE at Cal Poly - San Luis Obispo. In all three studies, two sections of the same course were taught by the same instructor. Approximately seventy students were divided between two sections. One section was randomly selected as the control group, and the other as the experimental group. Both sections were given weekly labs that contained identical content. The only difference was that the experimental group completed the labs in WebIDE, using the lock-step framework and incremental feedback. The control group used traditional development environments (e.g. Eclipse, vim/gcc).

The first two studies were pilot studies conducted in Fall 2010 and Spring 2011 while WebIDE was still being developed and was still a bit unstable. Preliminary results reported in May 2011 [15] that the WebIDE group showed a significant improvement in performance when writing a simple Android application. Additionally, among students with some programming experience, the WebIDE group was more proficient in writing unit tests.

The most complete evaluation of WebIDE was conducted in Fall 2011 in an Introduction to Computing (CS0) [20] course that taught beginning programmers how to build Android apps. Seventy-two students participated in the study in two sections of the same course. The course contained nine labs. The first three labs used visual drag-and-drop programming environments (Scratch and App Inventor). The fourth, fifth, sixth, and eighth labs introduced Java programming concepts

(data types, variables, operations, methods, classes, selection, loops, arrays). The seventh and ninth labs introduced Android. Table I summarizes the topics covered in the nine labs. Labs four through nine are highlighted to indicate the use of WebIDE and focus of this study. A more thorough analysis of just the arrays lab was presented by Hilton et al [21].

A. Evaluation of Impacts on Student Programming Skills

Students were given a pre-study quiz to assess their programming skills. Table II presents the pre-study quiz questions. The WebIDE section had weaker programming skills coming into the course (see Java PreQuiz in Figure 5 and Table III) , and they performed worse on the initial Scratch and App Inventor labs (prior to WebIDE). Just prior to lab four, a coin was flipped to determine which section would be control/experimental, without having analyzed results on the Java pre-quiz or the first three labs. All students were given the same lectures by the same instructor. The lectures included instruction and exercises on Java, Android, and using the Eclipse IDE. Although there may be better introductory development environments, Eclipse was chosen because of its strong support for Android development which all students would need in their team projects. The WebIDE section students then completed their labs using WebIDE. The control group completed identical labs using Eclipse. The control group was provided with shell programs that contained comments indicating where the students were to enter their code as instructed in the labs.

Labs four through eight included online quizzes (using Blackboard http://www.blackboard.com/) at the end of closed lab sessions. Figure 5 shows the average percentage score on the assessments. Figure 6 visualizes this same data a bit differently, showing the percentage differences on these quiz assessments. Students who used WebIDE scored between 2.51% and 12.19% better on all assessments after WebIDE was introduced.

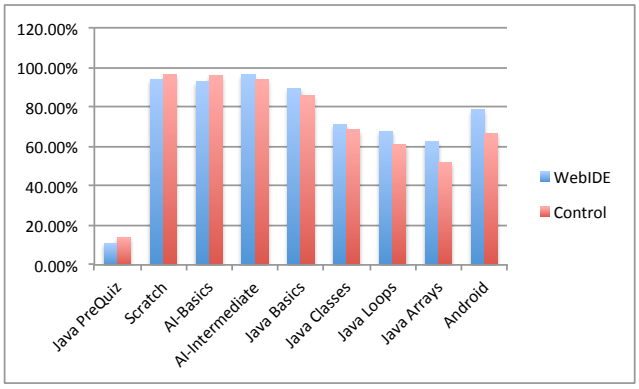

Fig. 5. Student Performance on Programming Assessments

Table III reports additional detail on student performance in the controlled experiment. In particular, scores are reported on computer programming assessments, demonstrating the weaker performance on the two pre-WebIDE assessments,

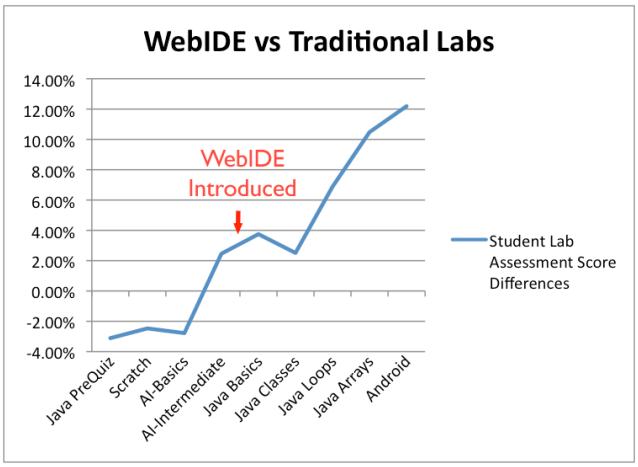

Fig. 6. Relative Student Performance on Programming Assessments

and consistently better performance after the introduction of WebIDE. Statistical significance was determined with $p < .05$ using a one-tailed univariate analysis of covariance test controlling for each student's score on the Java pre-study quiz.

B. Evaluation of Impacts on Student Opinions

Students were asked their opinions on the course, the labs, and their perspectives on their chosen computing major in a post-experiment on-line survey. Table IV reports these results based on a five point Likert scale with one being "Strongly Agree" and five being "Strongly Disagree." Smaller averages indicate more positive responses. The WebIDE students consistently were more positive about the labs and their major than the control group. Statistical significance was determined with $p < .05$ using a Two sample t-test. A Wilcoxon rank-sum test was also run on the first eight questions, identifying statistical significance with rows 2, 3, and 8.

Table V reports student responses to questions specifically about the IDE that they used in their labs (WebIDE or Eclipse). The numbers reported reflect the percent of students who answered 1-Strongly Agree or 2-Agree to the corresponding questions. Although the Eclipse users liked their IDE a bit more, the WebIDE users reported much stronger agreement on the positive affect of the IDE on their learning and its ability to provide helpful error messages. Based on student comments in free-response forms, there are still many improvements we can make to WebIDE labs. In particular, some students indicated frustration with the lock-step nature of the labs. If they got stuck, they couldn't go on until they got help (is this a good thing?). Other students wished that they could see the entire contents of their solution at the end of a lab, thus giving better context. This can be easily incorporated, and is under the purview of the lab author.

C. Threats to Validity

Like many academic studies, this evaluation is susceptible to several validity threats such as external factors that may have influenced students in the two groups (e.g. friendships formed

TABLE I
LAB TOPICS; * INDICATES USED WEBIDE

Lab #	Topics
1	Programming with Scratch, a visual programming environment for creating 2D animations, Control structures, event-driven programming, concurrency, variables.
2	Programming with App Inventor for Android, a visual programming environment for creating Android apps, Visual screen components (e.g. buttons, labels, images), click listeners, media components (e.g. audio), 2D drawing on a canvas.
3	More App Inventor for Android, Creating apps with multiple screens, integrating Google Maps, launching activities, creating procedures.
4 *	Java ints and operators, Built-in Java data types, corresponding operators, declaring variables, Control flow, selection control structures: if-then, if-then-else, switch, Conditions, comparison and boolean operators, Methods/functions, parameters, return types, Testing functions with examples and JUnit.
5 *	Java classes, Static methods, instance methods, constructors, Testing methods in classes with examples and JUnit.
6 *	Java looping control structures, Testing loops with examples and JUnit, While loops, for loops, nested loops, loops with nested selection.
7 *	Android basics, creating Hello World and Rock-Paper-Scissors game, Android project structure, manifest file, xml layout files, xml string files, Android Activity class, application lifecycle, onCreate method, attaching a layout to an Activity, Launching another Activity using intents, passing parameters to another Activity using Bundle.
8 *	Java arrays Declaring and initializing arrays, referencing array elements, out of bounds exception, Looping through arrays, passing arrays to methods as parameters, Arrays of objects, calling methods on objects in an array.
9 *	More Android, creating a Tic Tac Toe game, Designing a user interface, using a GridView layout, Designing an app with non-UI classes, keeping track of cell state and game state, implementing BaseAdapter.

TABLE II
PRE-STUDY QUIZ QUESTIONS

Question #	Question
1	Write a method/function that receives six parameters representing student scores in a course, and returns the student's weighted grade as a real number between 0 and 100. Each input is a real number between 0 and 100. The inputs/parameters are: labs, project1, project2, participation, midterm, final The weightings are: labs: 25%, project1: 15%, project2: 25%, participation: 10%, midterm: 10%, final: 15%
2	Write a Java class that represents a mobile device. Each device has three attributes: a name, price, and screen width in inches. Include a constructor that accepts all three attributes. Include a function/method that returns the cost per inch (price/width).
3	Write a JUnit test that constructs one instance of the class defined above, and tests that the cost per inch method works correctly.

within the class), and differences in the meeting times of the two sections. One particular threat to validity should be noted. In the final Android Tic Tac Toe lab (Lab 9), a large number of students in the control group discovered and used the WebIDE lab when they ran into challenges with completing the lab in Eclipse. When questioned, all students reported that this was their first use of WebIDE.

D. Anecdote

In addition to the evaluated courses, WebIDE is currently being used in an on-line commercial course at Udemy.[1] Students from the Fall 2011 CS0 course were invited to enroll in the Udemy course for free while they were taking a Spring CS2 course which also used Java. Although the percentage of female students in the Fall 2011 CS0 course was only 19% (14 out of 72), 64% (9 out of 14) of them signed up for the Udemy course when it was offered as a free supplement to their CS2 course. This compares to only 38% (22 out of 58) of the male CS0 students who signed up for the Udemy course. It would appear that female students are more likely to take advantage of online resources like WebIDE.

[1] http://www.udemy.com/java-essentials-for-android/

VII. RELATED WORK

A variety of online systems support development and evaluation of programs in a variety of languages [13], [19], [2], [10], [1], [27], [18], [39], [32], [26]. These systems are intended for developers, and do not operate as automated tutors.

Separately, automated tutors exist for a variety of academic fields. Samples include Biology/Genetics [33], Mathematics [23], and Physics [38]. Many of these tools have been evaluated, with promising results. For instance, Warnakulasooriya et al.[37] reports that their web-based automated Physics tutor improves student time to completion, reduces the need for hints, and improves the number of correct answers all by approximately 15%.

Not surprisingly, computing faculty and researchers have also built many software tools to support students as they learn

TABLE III
STUDENT PERFORMANCE ON PROGRAMMING ASSESSMENTS

Assessment	Total Possible	WebIDE Mean	Control Mean	Significant?
Java Pre-Study Quiz	25	2.72	3.50	No
Midterm Exam: App Inventor	20	14.56	14.92	No
Lab 4 Quiz: Java Basics, If, Methods	20	17.89	17.14	Yes
Midterm Exam: Java if	15	13.47	13.31	No
Midterm Exam: Java methods	20	17.89	16.50	Yes
Lab 5 Quiz: Java Classes	26	18.43	17.78	No
Lab 6 Quiz: Java Loops	12	8.11	7.29	Yes
Lab 7 Quiz: Android Basics	10	7.89	6.67	Yes
Lab 8 Quiz: Java Arrays	22	13.75	11.44	Yes
Comp. Lab Quiz: Java Arrays/Loops	12	10.17	8.75	Yes
Comp. Lab Quiz: Android	14	6.47	5.08	Yes
Final Exam	210	180.81	173.17	Yes
Final Exam: Java Class	12	11.08	10.08	Yes
Final Exam: JUnit Test	6	4.72	4.50	No
Final Exam: Android XML	8	7.47	7.36	No
Final Exam: Android Activity	12	8.83	7.97	No

TABLE IV
STUDENT OPINIONS

Question	WebIDE Mean	Control Mean	Significant?
Confident will complete major	1.28	1.61	Yes
Course stimulated interest	1.66	1.92	No
Computing is exciting	1.59	2.00	Yes
I liked this course	1.59	2.03	Yes
I am comfortable with Java	2.38	2.47	No
I am comfortable with Android	3.03	3.25	No
Labs helped learn Java	2.21	2.64	Yes
Labs helped learn Android	2.09	2.72	Yes
Liked Lab 4	2.03	2.31	No
Liked Lab 5	2.34	2.56	No
Liked Lab 6	2.22	2.36	No
Liked Lab 7	2.66	3.44	Yes
Liked Lab 8	2.34	2.81	Yes
Liked Lab 9	1.44	1.72	No

TABLE V
STUDENT OPINIONS OF IDE

Question	WebIDE Percent Positive	Control Percent Positive
I liked my IDE	65.7%	69.4%
My IDE helped me learn Java	84.4%	69.5%
My IDE helped me learn Android	68.8%	58.3%
My IDE's failure messages were helpful	59.4%	44.4%
My IDE's failure messages were confusing	62.5%	66.7%

to program. Valentine[36] reports that 22% of the CS1/CS2-related SIGCSE conference papers from 1984 to 2003 included software tools to aid learning. Some of the more popular tools include visualizations[22], Karel micro-worlds[8], [7], automated assessment tools[14], and pedagogical development environments such as DrRacket[17], Alice [12], and Scratch [31].

A few systems closely relate to WebIDE. TuringsCraft [4] is a commercial web-based system that presents interactive exercises for Python, C, C++, and Java. Truong et al.[35] created ELP [34] which provides fill-in-the-blank style exercises. Unlike WebIDE, TuringsCraft and ELP do not apply a TDL approach, and the exercises cannot be contributed or extended by individual faculty. A system by Azalov [5] allows instructors to parameterize code in order to generate programs chosen from similar families, but does not focus on evaluation or scripting of the tutor. CodeAcademy[11] provides interactive lessons for Javascript, and Microsoft's Pex4Fun[29] integrates "coding duels" for C#, Visual Basic, and F#.

Aleven et al.'s work with CTAT [3] highlights the common struggle with reducing the effort to author labs for intelligent tutoring systems. Parlante's CodingBat [30] adopts a test-based approach, although students do not write tests and the system is limited to a set of small, focused exercises. Edwards' Web-CAT[16] web-based automated grading tool assumes student

creation of automated (presumably test-driven) unit tests, but it provides no support for interactive labs.

VIII. Future Work

We are continuing to improve and use WebIDE. Current efforts are underway to improve the WebIDE user interface and to add some gamification aspects. The user base for WebIDE is expanding, and we look forward to seeing community contributions of labs and automated evaluators.

IX. Conclusions

WebIDE is unique in its combination of features: a TDL approach, completely web-based delivery, and intrinsic support for community-contributed content. WebIDE enables lab authors to create their own labs or use existing labs, and to give specific student feedback on individual exercises. WebIDE is flexible, scalable, and robust.

Classroom experiments indicate that students learn early programming concepts better with WebIDE than using traditional static labs with traditional IDE's. Furthermore, students report more positive experiences in learning to program with WebIDE over more traditional development environments.

Acknowledgments

The authors are grateful to Aaron Keen and Gene Fisher for their comments on an earlier version of this paper, and to Olga Dekhtyar for her assistance with the statistical analysis.

This material is based upon work supported by the National Science Foundation under Grant No. 0942488. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

We are also grateful to Amazon Web Services and Google for web services and lab development support.

References

[1] Go playground. http://golang.org/.

[2] Jsbin. http://jsbin.com/.

[3] V. Aleven, B. M. Mclaren, J. Sewall, and K. R. Koedinger. The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In *Proceedings ITS 2006*, pages 61–70. Springer-Verlag, 2006.

[4] D. Arnow and G. Weiss. Turingscraft. http://www.turingscraft.com/.

[5] P. Azalov. A web-based environment for introductory programming courses. *J. Comput. Small Coll.*, 22(4):260–267, 2007.

[6] K. Beck. *Test Driven Development: By Example*. Addison Wesley, November 2002.

[7] B. W. Becker. Teaching CS1 with karel the robot in Java. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 50–54, New York, NY, USA, 2001. ACM.

[8] J. Bergin, J. Roberts, R. Pattis, and M. Stehlik. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[9] J. Clements and D. Janzen. Overcoming obstacles to test-driven learning on day one. In *ICSTW '10: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 448–453, Washington, DC, USA, 2010. IEEE Computer Society.

[10] I. Cloud9 IDE. Cloud9. http://www.cloud9ide.com/.

[11] CodeAcademy. Codeacademy. http://www.codeacademy.com/.

[12] S. Cooper, W. Dann, and R. Pausch. Teaching objects-first in introductory computer science. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, SIGCSE '03, pages 191–195, New York, NY, USA, 2003. ACM.

[13] R. Data. W3 schools. http://www.w3schools.com/js/.

[14] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3):4, 2005.

[15] T. Dvornik, D. S. Janzen, J. Clements, and O. Dekhtyar. Supporting introductory test-driven labs with WebIDE. In *CSEE&T*, pages 51–60, 2011.

[16] S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3):1, 2003.

[17] R. B. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen. Drscheme: A programming environment for Scheme. *Journal of Functional Programming*, 12(2):159–182, 2002.

[18] D.-E. K. Gilad Khen and A. Weiss. Coderun. http://www.coderun.com/.

[19] Google. Google API playground. http://interactivesampler.appspot.com/.

[20] M. Haungs, C. Clark, J. Clements, and D. Janzen. Improving first-year success and retention through interest-based CS0 courses. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 589–594, New York, NY, USA, 2012. ACM.

[21] M. Hilton and D. S. Janzen. On teaching arrays with test-driven learning in WebIDE. In *Proceedings of the 17th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE'12, 2012.

[22] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259 – 290, 2002.

[23] W. P. Institute and C. M. University. Assistments. http://www.assistments.org/.

[24] D. Janzen and H. Saiedian. Test-driven learning: intrinsic integration of testing into the CS/SE curriculum. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, page 258. ACM, 2006.

[25] D. Janzen and H. Saiedian. Test-driven learning in early programming courses. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, volume 40, pages 532–536. ACM, 2008.

[26] R. Krahn, D. Ingalls, R. Hirschfeld, J. Lincke, and K. Palacz. Lively wiki a development environment for creating and sharing active web content. In *WikiSym '09: Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, pages 1–10, New York, NY, USA, 2009. ACM.

[27] S. R. Labs. ideone. http://ideone.com/.

[28] T. O. S. S. C. Ltd. Jing. http://www.thaiopensource.com/relaxng/jing.html.

[29] Microsoft Research. Pex4fun. http://pex4fun.com/.

[30] N. Parlante. Codingbat. http://codingbat.net.

[31] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: programming for all. *Commun. ACM*, 52:60–67, Nov. 2009.

[32] ShiftEdit. Shiftedit. http://shiftedit.net/.

[33] The Concord Consortium. Biologica. http://biologica.concord.org/.

[34] N. Truong, P. Bancroft, and P. Roe. Elp. http://www.elp.fit.qut.edu.au/.

[35] N. Truong, P. Bancroft, and P. Roe. Learning to program through the web. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 9–13, New York, NY, USA, 2005. ACM.

[36] D. W. Valentine. CS educational research: a meta-analysis of SIGCSE technical symposium proceedings. *SIGCSE Bull.*, 36(1):255–259, 2004.

[37] R. Warnakulasooriya, D. Palazzo, and D. E. Pritchard. Evidence of problem-solving transfer in web-based socratic tutor. In *Physics Education Research Conference 2005*, volume 818 of *PER Conference*, pages 41–44, Salt Lake City, Utah, August 10-11 2005.

[38] R. Warnakulasooriya and D. E. Pritchard. Mastering physics. http://www.masteringphysics.com/.

[39] D. Yoo, B. Hickey, E. Schanzer, and S. Krishnamurthi. WeScheme. http://www.wescheme.org/.